Full Guide to understand Docker

Presentation · June 2025

CITATIONS
0

READS
84

1 author:

Youcef Benabderrezak
University of Boumerdes
241 PUBLICATIONS **35** CITATIONS

SEE PROFILE

Part 1 : Install Docker on Linux Mint

Remove Old Versions of Docker

sudo apt-get remove docker docker-engine docker.io containerd runc

- Older Docker versions might still be installed.
- This command makes sure all outdated Docker components are fully removed to prevent system conflicts during the new installation.

Update the System Package Index

sudo apt-get update

- The system uses a local "package list" to know where to find software.
- This command updates that list to ensure you install the latest packages and security patches.

Install Required Software

sudo apt-get install apt-transport-https ca-certificates curl software-properties-common

- These packages :
  - **apt-transport-https** : Allows downloading packages securely over HTTPS.
  - **ca-certificates** : Provides security certificates to ensure trusted connections.
  - **curl** : A tool to transfer data from or to a server (used to fetch Docker's GPG key).
  - **software-properties-common** : Helps manage external software repositories.

Add Docker's Official GPG Key

curl -fsSL https ://download.docker.com/linux/ubuntu/gpg | sudo apt-key add -

- GPG keys are security signatures that ensure the software you download is from the official Docker team and hasn't been modified by attackers.

Add Docker Repository

sudo add-apt-repository "deb [arch=amd64] https ://download.docker.com/linux/ubuntu focal stable"

- Docker isn't part of the Linux Mint default package list.
- This command connects your system to Docker's official software source so you can download Docker directly from them.

Install Docker

sudo apt-get update

sudo apt-get install docker-ce docker-ce-cli containerd.io

- **docker-ce** : The Docker Community Edition (the actual Docker engine).
- **docker-ce-cli** : The command-line interface for Docker.
- **containerd.io** : A core part that manages container life cycles.

Verify Docker Installation

docker --version

- This checks if Docker was successfully installed and shows the version you are using.

Start Docker and Enable It to Start on Boot

sudo systemctl start docker

sudo systemctl enable docker

- systemctl start docker : Starts the Docker engine right now.
- systemctl enable docker : Configures Docker to start automatically when you turn on your computer.

Run Docker Without 'sudo'

sudo usermod -aG docker $USER

- Normally, Docker needs sudo because it controls system-level processes.
- This command gives your user permission to run Docker without sudo.
- **Important :** You must log out and log back in for this to take effect.

Part 2 : Docker Key Concepts (Detailed)

- **Docker :** A platform that allows you to package software with everything it needs (like libraries, system tools, and dependencies) into small units called containers
- **Docker Image :**
  - A pre-made, read-only "blueprint" for creating containers.
  - *Images contain the application and its environment.*
- **Docker Container :**
  - A running, isolated instance of an image.
  - *Containers share your system's OS but run like independent systems.*
- **Dockerfile :**
  - A simple text file with instructions for building a Docker image.
  - *You write steps like : "Start with Ubuntu, install Nginx, copy files, etc."*
- **Docker Hub :**
  - A public repository where you can download popular Docker images (like Ubuntu, MySQL, Nginx) or upload your own.
  - *It's like GitHub, but for Docker images.*

Part 3 : Essential Docker Commands

- docker --version :

  Displays the Docker version installed.

  Confirms Docker is working.

- docker info :

  Shows full system details like the number of containers, images, storage drivers, and system configuration.

  Useful to verify Docker is correctly set up.

- docker run <image> :

  Creates and starts a container from an image.

  Example : docker run ubuntu will start an Ubuntu container.

- docker ps :

  Lists all running containers.

  Shows container ID, image, status, ports, etc.

- docker ps -a :

  Lists all containers (both running and stopped).

  Good for finding and cleaning old containers.

- docker images :

  Lists all images downloaded to your system.

  You can delete unused images later to free space.

- docker stop <container_id> :

  Stops a running container safely.

- docker rm <container_id> :

  Deletes a container that you don't need anymore.

- docker rmi <image_id> :

  Deletes an image from your system.

  Use this when you want to free disk space.

Part 4 : Hands-On Docker Exercises (Detailed)

Exercise 1 : Run the Hello World Container

docker run hello-world

- Docker downloads a small image that simply prints : "Hello from Docker!"
- *Confirms Docker works and can pull images from Docker Hub.*

Exercise 2 : Run Ubuntu in Interactive Mode

docker run -it ubuntu

- -it : Runs the container in interactive terminal mode.
- This gives you a real terminal inside the Ubuntu container.
- *Try commands like ls, pwd, apt-get update inside the container.*

⇒ To exit : **exit**

Exercise 3 : Run an Nginx Web Server

docker run -d -p 8080 :80 nginx

- -d : Runs the container in "detached" (background) mode.
- -p 8080 :80 : Maps port 80 inside the container to port 8080 on your machine.
- *Access it in your browser : http ://localhost :8080 and see the Nginx welcome page.*

Exercise 4 : Manage Containers

- List running containers : **docker ps**
- Stop a container : **docker stop <container_id>**
- Remove a container : **docker rm <container_id>**
- *Each container has a unique ID, use it for stop/remove operations.*

Exercise 5 : Build a Custom Docker Image

Step 1 : Create a Dockerfile

FROM ubuntu

RUN apt-get update && apt-get install -y nginx

CMD ["nginx", "-g", "daemon off;"]

- FROM ubuntu : Start from a clean Ubuntu image.

- RUN apt-get ... : Install Nginx.
- CMD ... : Start Nginx in the foreground (container will keep running).

Step 2 : Build the Image

docker build -t my-nginx .

- -t my-nginx : Names the image "my-nginx".
- . : The current directory (must contain the Dockerfile).

Step 3 : Run the Custom Image

docker run -d -p 8081 :80 my-nginx

- *Open your browser : http ://localhost :8081 to see your custom web server running!*

📚 **Part 5 : Extra Useful Docker Commands**

See container logs :

docker logs <container_id>

- *View what's happening inside the container (errors, startup messages, etc.).*

List Docker networks :

docker network ls

- *Shows how Docker containers communicate.*

Enter a Running Container :

docker exec -it <container_id> bash

- *Allows you to directly interact inside a running container's terminal.*

Clean Up System :

docker system prune

- *Removes stopped containers, old images, and unused networks to free up space.*

🎯 Suggested Projects to Advance Your Skills

- Run a **MySQL container** and connect to it.
- Run **PHP and MySQL containers together** using Docker Compose.
- Build a **Node.js app** inside a Docker container.
- Run a **Vue.js or React app** using Nginx as a static file server.
- Create a **multi-container project** with frontend, backend, and database.

Full Guide: Running a PHP Web App with Docker on Linux Mint

This will teach you:

1. How to **run Apache and PHP** using Docker.
2. How to **connect PHP to MySQL using Docker Compose.**
3. How to **structure your web project to run fully inside Docker.**

📦 Step 1: Prepare Your Project Structure

Let's create a minimal PHP web app that connects to MySQL.

my-php-app/

│

├── src/

│ ├── index.php

├── docker-compose.yml

├── Dockerfile

└── .dockerignore

✍️ Step 2: Write the PHP File

Create src/index.php:

<?php

$host = 'mysql';

$db = 'mydb';

$user = 'root';

$pass = 'root';

```
try {
    $pdo = new PDO("mysql:host=$host;dbname=$db", $user, $pass);
    echo " Connected to MySQL successfully!<br>";
    // Create a table if not exists
     $pdo->exec("CREATE TABLE IF NOT EXISTS messages (id INT AUTO_INCREMENT PRIMARY KEY, message VARCHAR(255))");
    // Insert a new message
    $pdo->exec("INSERT INTO messages (message) VALUES ('Hello from Docker!')");
    // Read messages
    $stmt = $pdo->query("SELECT * FROM messages");
    while ($row = $stmt->fetch()) {
        echo "📝 " . $row['message'] . "<br>";
    }
} catch (PDOException $e) {
    echo "❌ Connection failed: " . $e->getMessage();
}
?>
```

This script connects to MySQL, creates a table, inserts a message, and displays it.

Step 3: Write the Dockerfile for PHP + Apache

Create a file named Dockerfile:

```
FROM php:8.2-apache
```

```
# Install PDO MySQL extension

RUN docker-php-ext-install pdo pdo_mysql

# Copy your PHP files to Apache server root

COPY src/ /var/www/html/

# Expose Apache port

EXPOSE 80
```

⇒ **This builds an Apache server with PHP 8.2 and MySQL support, and copies your PHP code into the web server directory**

Step 4 : Write the Docker Compose File

Create docker-compose.yml:

```
version: '3.8'

services:

 php-apache:

   build: .

   ports:

     - "8080:80"

   volumes:

     - ./src:/var/www/html

   depends_on:

     - mysql
```

```
  mysql:
    image: mysql:5.7
    environment:
      MYSQL_ROOT_PASSWORD: root
      MYSQL_DATABASE: mydb
    ports:
      - "3306:3306"
    volumes:
      - mysql-data:/var/lib/mysql
volumes:
  mysql-data:
```

This file:

- Builds the PHP+Apache container.
- Pulls a MySQL image.
- Connects the PHP container to the MySQL container using Docker's internal network.
- Maps your code directly to the running container for live updates.

🚀 Step 5: Start the Web App

In the terminal:

```
docker-compose up --build
```

- --build: Forces Docker to rebuild the PHP image.

- Docker will:
    - Build the PHP+Apache image.
    - Download the MySQL image.
    - Start both containers in a connected network.

🌐 Step 6: Open Your App

In your browser:

http://localhost:8080

You should see :

- A success message showing connection to MySQL.
- The list of inserted messages.

🛠️ Step 7: Manage Containers

Stop the containers : docker-compose down

Restart the containers : docker-compose up

View logs (real-time) : docker-compose logs -f

Enter the PHP container : docker exec -it <php_container_id> bash

Enter MySQL container :

docker exec -it <mysql_container_id> bash

mysql -u root -p

Enter password: root

Step 8: Important Docker Compose Concepts

- build: .

  Docker Compose will build the image using the Dockerfile in the current directory.

- depends_on:

  Tells Docker to start the MySQL container before the PHP container.

- volumes:

  Connects your project files directly into the container (live sync).

- environment:

  Used to pass MySQL configuration like passwords and database names.

Extra: Best Practices

- Use .dockerignore to avoid copying unnecessary files into the Docker image.
- Separate php.ini files if you want to customize PHP settings.
- Use phpmyadmin as an additional container to visually manage your MySQL database.

Example addition to docker-compose.yml:

```
phpmyadmin:
  image: phpmyadmin/phpmyadmin
  ports:
    - "8081:80"
  environment:
    PMA_HOST: mysql
```

👉 Access it at: http://localhost:8081

🎯 Next Step :

- Add **phpMyAdmin configuration**
- Build a **production version with persistent storage and networks**
- Build a **scientific paper on container-based PHP deployments**